




**School of Computing Technologies COSC1114****Operating Systems Principles Project 1 –****Multithreading & Synchronisation**

	<b>Assessment Type:</b> Group project (maximum of four students) Submit online via <a href="#">Canvas</a> → <a href="#">Assignments</a> → <a href="#">Assignment 2</a> Clarifications/updates may be made via announcements and relevant discussion forums.
	<b>Due Date:</b> Monday, September 8th, 2025, at 23:59.
	<b>Weighting:</b> 30 marks that contributes 30% of the total assessment.

**1. Overview**

This assignment focuses on the concept of multithreading. In this assignment, you will implement two tasks to investigate how multithreading enables parallel computing and the challenges that must be managed to ensure consistency.

**2. Learning outcomes**

This assessment is relevant to the Course Learning Outcomes CLOs 2, 5, and 6.

**3. Assessment details**

This assessment will determine your ability to

1. Understand the concepts taught over the first 5 weeks of the course.
2. Work independently in self-directed study or collaboratively with your teammate to research the identified issues.

**4. Academic integrity and plagiarism**

It is understood by us that many of the algorithms used in this course have common implementations. You are welcome to look at online code examples to understand possible solutions to the set problems. However, what you submit must be your own work and your submission will be checked and compared with other solutions.

Submitting material generated by an AI tool as your own work constitutes plagiarism and academic dishonesty. **DO NOT** simply copy other people's work, it is not difficult for us to detect copied work, and we will pursue such cases.

**5. Group work**

This assignment can be done in a team of up to four students. Teams are expected to collaborate on all aspects of the task, rather than dividing it into independent parts. You may want to brainstorm together, draft solutions jointly, and review each other's work at every stage to ensure mutual understanding and shared responsibility.

Should one team member be unable to complete their contribution by the deadline for any reason, the remaining team member(s) is still responsible for submitting the assignment on time.

Students collaborating in team must include their work log (date, student name, tasks completed, and hours spent) as the final page of their PDF submission, to assist markers in resolving any potential disputes.

Please note that extensions and special consideration requests are individual. To apply, you will need to withdraw from the group and do the assignment *\*individually\**. That is, if you want an individual extension or special consideration, you must do the assignment individually. A group extension or special consideration may be granted if all group members are impacted, and this needs to be approved by Course Coordinator.

To form a team, go to [Canvas → People](#), then select [Project 1 group](#) tab. From there, you can drag and drop your name from the list on the left into an available group on the right. If you have any trouble forming a team this way, please seek help from a teaching staff member during your tutorial session.

## 6. Submission

Your assignment should submit via [Canvas → Assignments → Project 1](#).

The submission should contain **two** parts, a **.zip file** and a **.txt file**.

The details of .zip file are below:

1. The source code (in C++ or C) of all the tasks and subtasks. **If you use any programming language other than C++ or C for the implementation, you will receive a 0 for this assignment.**
2. **A README file** that specifies exactly **which** tasks and subtasks you've completed and includes instructions on **how** to run your code on any of the Core Teaching Servers. They are  
**jupiter.csit.rmit.edu.au**, or  
**saturn.csit.rmit.edu.au**, or  
**titan.csit.rmit.edu.au**,
3. You also need to explicitly describe where you have implemented locks and condition variables, including the file names and line numbers in the README file.  
Please include **all** files mentioned above in a zip file. You can submit via [Canvas → Assignments → Project 1](#). The file name can be your student number (if you complete it individually) or your team number (if you complete it within a team). E.g., **s1234567.zip** or **project1\_group1.zip**.

Besides the zip file, organize the source code of tasks in a separate text file s1234567.txt or project1\_group1.txt. (copy & paste them into a text editor and then save it). This text file is for Turnitin plagiarism check.

Please submit the **.txt file** and the **.zip file** together. (You can add an additional file by clicking "Add Another File" when you are submitting.)

**Your submission will not be graded if you fail to submit your txt or if your txt does not pass the Turnitin plagiarism check.**

## 7. Late submission policy

A penalty of 10% per day of the total available marks will apply for each day being late. After 10 days, you will receive zero marks for the assignment.

If you want to seek an extension of time for assignment submission, you must have a substantial reason for that, such as unexpected circumstances. **Reasons such as, unable to cope with study load, is not substantial.** Also, you must apply for an extension as soon as possible. Last minute extensions cannot be granted unless it attracts special consideration.

Please find out how to apply for special consideration via this [link](#).

Any student wishing an extension must go through the official procedure for applying for extensions and must apply at least a week before the due date. Do not wait till the submission due date to apply

for an extension.

## 8. Assignment tasks

You need to decide on the program's structure, like whether to use header files, a single .cpp file, or separate files for all utility functions. However, you should have at least two files: one for **task 1** and another for **task 2**. You can name them "**mmcopier.cpp**" and "**mscopyer.cpp**", respectively.

### Task 1 – Multithreaded multiple file copying (5 marks)

In this task, each thread copies one file from a source to a destination. Threads run concurrently, each handling its own file copy independently.

Your program should be executed with the following command:

`./mmcopier n source_dir destination_dir`

where  $n$  ( $2 \leq n \leq 10$ ) is the number of files, which corresponds to the number of threads used, to be copied under `source_dir`, and `destination_dir` is the target directory.

For example, `./mmcopier 3 source_dir destination_dir` will copy **source1.txt**, **source2.txt**, and **source3.txt** under `source_dir` to `destination_dir`, using 3 threads.

Please find the sample `source_dir.zip` in the assignment page. The same folder/directory will be used for grading.

### Task 2 – Multithreaded single file copying (25 marks)

Write a multi-threaded program that reads data from one file and writes it to another.

#### Subtask 1: You will have a 'team' of readers and a 'team' of writer threads. (5 marks)

The reader threads will take turns reading lines from the source file and storing them in a shared queue. The writers likewise will take turns removing lines from the queue and writing them to the target file.

**Note: Reading and writing should occur in parallel.** The objective is to develop a shared queue, and you will need to determine the appropriate conditions, such as a shared queue that can accommodate a maximum of 20 lines.

We provide you with a bash script `generate_text.sh` to generate a source file (using a dictionary) for testing purposes. **Please find the bash script and the dictionary in the assignment page.**

#### Subtask 2: Insert locks (10 marks)

You will need to insert locks into the appropriate places using the `pthread_mutex` API to restrict other threads from accessing critical sections, thereby avoiding race conditions and deadlocks.

#### Subtask 3: Avoid busy waiting (10 marks)

You are to avoid busy waiting. Busy waiting is when a thread does nothing but check if some condition is met repeatedly. Instead, when you are waiting for a condition to be met you should use a `sleep()` function call or if you are aiming for a high distinction, a condition variable (see The API section for details). What this means is that if you use the `sleep()` solution you can get half the marks here but only a solution using the `pthread_cond` API can attract full marks.

Your program will be invoked as:

`./mscopyer n source_file destination_file`

where  $n$  will be an integer between 2 and 10, `source_file` is the file to be copied, and `destination_file` is the destination (these maybe be any valid filesystem path).

For example: `./mscopyer 10 input output` will use 10 readers and 10 writers to copy the file `input` to the file `output`.

Hints: the readers must wait when the shared queue is full, and the writers must wait when the shared queue is empty.

## 9. Technical Specifications and Technical Support

### Build Environment

Please note that it is expected that your program will compile and run cleanly on the provided servers:

- titan.csit.rmit.edu.au

- [jupiter.csit.rmit.edu.au](http://jupiter.csit.rmit.edu.au)
- [saturn.csit.rmit.edu.au](http://saturn.csit.rmit.edu.au)

A 20% penalty will be applied if the markers are unable to run your program on these servers.

### Code Quality

Please note that while code quality is not specifically marked for, it is part of what is assessed as this course teaches operating systems principles partly through the software you develop. As such lack of comments, lack of error checking, good variable naming, avoidance of magic numbers, etc., may be taken into consideration by your marker in assigning marks for the components of this assignment, although operating systems principles do take priority.

### Memory Correctness

All accesses to memory are expected to be correct (no reading from uninitialized memory, out of bounds access, not freeing memory that was not allocated) and we will check for these things with [valgrind](#) on the provided UNIX servers using the following command:

`valgrind --track-origins=yes --leak-check=full --show-leak-kinds=all executable_name args`

Please check your code prior to submission with [valgrind](#). You may want to ask question during the tutorial session if you are new to [valgrind](#).

**Memory incorrect code will attract a deduction especially as this is an operating systems course. A 10% penalty will be imposed if the program exhibits memory leaks.**

### Implementation and Compilation

Programming Language: Implement the system in C++ or C.

Concurrency Constructs: Use POSIX threads. In other words, the [pthread](#) library.

To compile C++/C program with [pthread.h](#) library, you must append `-lpthread` after the compile command. For example,

**`gcc -Wall -Werror -o mscopier mscopier.c -lpthread`**

This command will tell the compiler to execute program with `pthread.h` library.

Error Handling: Your program must gracefully handle and report errors such as failed memory allocation or file access issues.

**Makefile:** Create a makefile to compile your programs.

- `make all` will build two programs `mmcopier` and `mscopier`, at once.
- `make clean` will get rid of object and executable files.
- Make sure you compile with the following flags and fix the errors before submitting your assignment: `-Wall -Werror`

**Please seek help in the tutorial session if you're new to Makefile.**

## 10. Rubric and marking guidelines

- The rubric is in [Canvas → Assignments → Project 1](#) (At bottom of the page)
- For each assessment requirement, the rubric is designed to be general and the details of what you can improve on will be outlined by your marker in the comments. Nevertheless, this should be treated as a guide to what to aim for.
- Here are five bins as a general reference, and your mark may land in between,
  1. Little to no attempt (0%): Not attempted.
  2. Poor (25%): Some attempt but did not get the code working or does not compile.
  3. OK (50%): Code compiles but threading is not correctly implemented.
  4. Good (75%): Threading works but there are minor problems.
  5. Excellent (100%): Well-done.
- Please note that code with segmentation faults in the respective part of the assignment we are assessing cannot get above 50%, and code with problems highlighted by [valgrind](#) cannot get over 90% in that part of the assignment.

## 11. The API

Note: this is a basic API of the functions we want you to use for concurrency and thread safety. The

man pages have examples of how to use these functions.

**Please note that you are expected to check the return values of all functions.**

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,  
    void *(*start_routine)(void*), void *arg);
```

The `pthread_create()` function creates a thread with the specified attributes. The new thread starts execution by invoking `start_routine()`; `arg` is passed as the sole argument of `start_routine()`.

**Hint:** a void pointer is just a pointer that can point to any pointer type. `attr` can be ignored for this assignment (just pass in a `nullptr`, the default attributes shall be used).

Find out more on [titan](#): `$man pthread_create`

Your program should wait for the threading function to complete by calling the function:

```
int pthread_join(pthread_t thread, void **retval);
```

The `pthread_join()` function waits for the thread specified by `thread` to terminate. If that thread has already terminated, then `pthread_join()` returns immediately. The thread specified by `thread` must be joinable.

Find out more on [titan](#): `$man pthread_join`

The following functions are used to manipulate mutexes which are variables that enforce mutual exclusion. A mutex can only be locked by one thread at a time and all other threads that try to lock that mutex will sleep until the mutex becomes available.

```
int pthread_mutex_init(  
    pthread_mutex_t *mutex,  
    const pthread_mutexattr_t *attr);
```

The `pthread_mutex_init()` function shall initialize the mutex referenced by `mutex` with attributes specified by `attr`.

**Hint:** you can pass in `nullptr` for `attr` and the default attributes are used.

Find out more on [titan](#): `$man pthread_mutex_init`

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

The `pthread_mutex_destroy()` function shall destroy the mutex object referenced by `mutex`; the mutex object becomes, in effect, uninitialized.

Find out more on [titan](#): `$man pthread_mutex_destroy`